# 题目

给定一个单词列表和两个单词 word1 和 word2，返回列表中这两个单词之间的最短距离。

**示例:**

假设 words = ["practice", "makes", "perfect", "coding", "makes"]

输入: word1 = “coding”, word2 = “practice”

输出: 3

输入: word1 = "makes", word2 = "coding"

输出: 1

**注意:**

你可以假设 word1 不等于 word2, 并且 word1 和 word2 都在列表里。

# 分析

## 方法一：暴力

算法

一个比较简单的方法是遍历整个数组直到找到第一个单词。每次我们找到跟第一个单词一样的词时，我们就遍历整个数组去找跟第二个单词一样的词，并求解最近距离。

public int shortestDistance(String[] words, String word1, String word2) {

int minDistance = words.length;

for (int i = 0; i < words.length; i++) {

if (words[i].equals(word1)) {

for (int j = 0; j < words.length; j++) {

if (words[j].equals(word2)) {

minDistance = Math.min(minDistance, Math.abs(i - j));

}

}

}

}

return minDistance;

}

复杂度分析

时间复杂度：O(n^2) ，这是因为每次找到一个 word1 ，我们都需要遍历一遍整个数组去找 word2 出现的位置。

空间复杂度：O(1) 。没有使用额外的空间。

## 方法二：遍历一次

算法

我们可以记录两个下标 i1 和 i2 来显著提高暴力的时间复杂度，我们保存 word1 和 word2 的 最近 出现位置。每次我们发现一个新的单词出现位置，我们不需要遍历整个数组去找到另一个单词，因为我们已经记录了最近出现位置的下标。

public int shortestDistance(String[] words, String word1, String word2) {

int i1 = -1, i2 = -1;

int minDistance = words.length;

int currentDistance;

for (int i = 0; i < words.length; i++) {

if (words[i].equals(word1)) {

i1 = i;

} else if (words[i].equals(word2)) {

i2 = i;

}

if (i1 != -1 && i2 != -1) {

minDistance = Math.min(minDistance, Math.abs(i1 - i2));

}

}

return minDistance;

}

复杂度分析

时间复杂度：O(n)。这个问题的解法是线性的，我们无法比O(n)更快是因为我们至少要遍历每个元素一次。

空间复杂度：O(1) 。没有使用额外空间。

## 方法三：哈希

解法1：用哈希表收集每个单词到其所有可能下标数组的映射。然后获取两个单词word1和Word2的下标数组，二重循环任何一对组合，计算最短距离即可。O(N\*N)

解法2：其实不用两次遍历，一次即可。用下标idx1表示上一个word1的位置，用下标idx2表示上一个word2的位置，如果两者皆不为-1，则更新最短距离，O(N)时间既可以算出最短距离。

解法3：其实不用两个变量idx1和idx2，一个idx即可，表示上一个word1或者word2的下标，当idx存在（不为-1）且当前单词i为word1或者word2且跟上一个wordsDict[idx]处的单词不相同时，更新最短距离。也是O(N)时间。

class Solution1 {

public:

int shortestDistance(vector<string>& wordsDict, string word1, string word2) {

unordered\_map<string, vector<int>> mp;

for (int i = 0; i < wordsDict.size(); i++) {

mp[wordsDict[i]].push\_back(i);

}

int minDist = numeric\_limits<int>::max();

for (const auto& idx1 : mp[word1]) {

for (const auto& idx2 : mp[word2]) {

minDist = min(minDist, abs(idx1 - idx2));

}

}

return minDist;

}

};

class Solution2 {

public:

int shortestDistance(vector<string>& wordsDict, string word1, string word2) {

int idx1 = -1;

int idx2 = -1;

int minDist = numeric\_limits<int>::max();

for (int i = 0; i < wordsDict.size(); i++) {

if (wordsDict[i] == word1) {

idx1 = i;

} else if (wordsDict[i] == word2) {

idx2 = i;

}

if (idx1 != -1 && idx2 != -1) {

minDist = min(minDist, abs(idx1 - idx2));

}

}

return minDist;

}

};

class Solution {

public:

int shortestDistance(vector<string>& wordsDict, string word1, string word2) {

int idx = -1;

int minDist = numeric\_limits<int>::max();

for (int i = 0; i < wordsDict.size(); i++) {

if (wordsDict[i] == word1 || wordsDict[i] == word2) {

if (idx != -1 && wordsDict[i] != wordsDict[idx]) {

minDist = min(minDist, abs(i - idx));

}

idx = i;

}

}

return minDist;

}

};